Санкт-Петербургский государственный университет
Математическое обеспечение и администрирование информационных систем

Параллельное программирование

Митенева Виктория Андреевна

Распараллеливание решения одномерной краевой задачи

Выпускная квалификационная работа

Научный руководитель:
д. ф.-м. н., профессор Бурова И. Г.

Рецензент:
ст. преп. Мирошниченко И. Д.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY
Software and Administration of Information Systems

Parallel Programming

Viktoriia Miteneva

Paralleling solutions of one-dimensional boundary-value problem

Graduation Project

Scientific supervisor:
professor Irina Burova

Reviewer:
senior teacher Irina Miroshnichenko

Saint-Petersburg
2017

Оглавление

Введение

Дифференциальными уравнениями описываются многие процессы в технике, химии, экономике, биологии, психологии и т.д. С помощью диффернециальных уравнений можно смоделировать большое количество физических задач. Чаще всего эти методы имеют ограниченное применение, или настолько сложны, что для упрощения решения используют методы приближенного численного решения. Математические модели реальных процессов иногда получаются довольно сложными, и могут не иметь аналитического решения. В данном случае используется решение с использованием приближенной модели или же приближенных (численных) методов. Внедрение в научную деятельность современных ЭВМ, а также их вычислительная мощность дало возможность решать трудные уравнения, достаточно точно описывающие исследуемые явления, а также моделировать всевозможные системы.

С середины 80-х годов оживленно развивается теория краевых задач для линейных дифференциальных уравнений 2 порядка с сильным вырождением. Этот тип задач поевляется при построении математических моделей ряда физических процессов, иследованием которых занимаются такие области науки, как физика плазмы и газового разряда, ядерная физика. В нелинейной оптике исследуется задача о самофокусировке лазерного луча. В данной задаче сутью является то, что из-за качеств среды световой пучок собирается в точку и плотность энергии в данной точке становиться бесконечной. Точное решение задачи не найдено и асимптотика поведения решения неизвестна. Определение точного решения вышеописанной задачи и задач, ей подобных, вероятно только в маленьком числе частных случаев. В остальных случаях обобщённое решение определить невозможно.В связи с этим, актуальным является изучение аналогичных задач в общей постановке, а еще создание эффективных действенных методов численного анализа.

1. Цель работы

Целью данной дипломной работы является разработка алгоритма и написание программы для решения краевой задачи вариационным методом. Для достижения этой цели в рамках работы были сформулированы следующие задачи.

- Изучить построение базисных сплайнов.
- Изучить методы решения краевой задачи вариационным методом.
- Разработать алгоритм решения краевой задачи вариационным методом.
- Составить последовательную программу и распараллелить ее.
- Получить результаты и провести анализ численного счета.

2. Обзор

2.1. О базисных сплайнах

Пусть функция $u \in C^2(R^1)$, $\{x_j\}$ – сетка упорядоченных узлов

$$a = x_0 < x_1 < \ldots < x_N = b.$$

Аппроксимацию $\widetilde{u}(x)$ функции u будем строить в виде [3]

$$\widetilde{u}(x) = \sum_j u(x_j)\omega_j(x), \tag{1}$$

где $\omega_j(x)$ – интерполяционный базис. [11] При построении сплайнов $\omega_j(x)$ предполагаем, что

$$\operatorname{supp} \omega_j(x) = [x_{j-1}, x_{j+1}].$$

Считая, что кратность накрытия произвольной точки x носителями базисных сплайнов $\omega_j(x)$ ограничена числом 2, нетрудно заметить, что в сумме выражения (1) небольшое количество слагаемых:

$$\widetilde{u}(x) = u(x_j)\omega_j(x) + u(x_{j+1})\omega_{j+1}(x). \tag{2}$$

И минимальные интерполяционные сплайны $\omega_j(x)$ представлены следующим образом:

$$\omega_j(x) = \begin{cases} \dfrac{x - x_{j-1}}{x_j - x_{j-1}}, x \in [x_{j-1}, x_j], \\ \dfrac{x - x_{j+1}}{x_j - x_{j+1}}, x \in [x_j, x_{j+1}], \\ 0, \quad x \notin [x_{j-1}, x_{j+1}] \end{cases} \tag{3}$$

Построенные таким образом сплайны $\omega_j(x)$ имеют следующие свойства:

1) $\omega_j(x_j) = 1$,

2) $u - \widetilde{u} \equiv 0$ для $u = 1, x$,

3) supp $\omega_j = [x_{j-1}, x_{j+1}]$,

4) $|u - \widetilde{u}| \leq h^2 C$, где $h = \max_j(x_{j+1} - x_j)$, $C = \text{const}$.

2.2. О вариационных методах

На промежутке (0, 1] будем решать следующую краевую задачу

$$(-k(x)u'(x))' + q(x)u(x) = f(x) : \quad u(1) = 0, \tag{1}$$

где

$$k(x) = x^\alpha a(x), \quad \alpha \in (1, 2),$$

$$a(x) > 0, \quad q(x) \geq 0, \quad f \in L_2.$$

Приближенное решение $\widetilde{u}$ будем искать в виде [3]

$$\widetilde{u}(x) = \sum_j c_j \omega_j(x), x \in [x_j, x_{j+1}] \tag{2}$$

где $\omega_j(x)$ – базисные сплайны [5], а c_j - некоторые параметры, определяемые из задачи поиска минимума функционала

$$I = \int_0^1 [k(x)u'^2(x) + q(x)u^2(x) - 2f(x)u(x)]dx. \tag{3}$$

Задача на минимум функционала (3) на пространстве функций вида (1) приводит к следующей системе уравнений [9]:

$$\sum_{i=1}^{N} c_i g_{ij} = f_j, \quad j = 1, \ldots, N, \tag{4}$$

где коэффициенты и правая часть вычисляются по формулам:

$$g_{ij} = \int_0^1 (k(x)\omega'_i(x)\omega'_j(x) + q(x)\omega_i(x)\omega_j(x))dx,$$

$$f_j = \int_0^1 f(x)\omega_j(x)dx.$$

3. Постановка задачи

Рассмотрим дифференциальное уравнение

$$(-x^{\alpha}u'(x))' + u(x) = f(x), \tag{1}$$

$$u(1) = 0,$$

на промежутке (0, 1], где $\alpha \in [1, 2]$.

Тогда при

$$f(x) = \frac{x^{3-\alpha} - 2(3-\alpha)x - 1}{3-\alpha} \tag{2}$$

точное решение u будет представлено в виде

$$u = \frac{x^{3-\alpha} - 1}{3-\alpha} \tag{3}$$

Приближенное решение $\widetilde{u}$ будем искать в виде

$$\widetilde{u}(x) = \sum_j c_j \omega_j(x), \quad x \in [x_j, x_{j+1}], \tag{4}$$

где $\omega_j(x)$ – базисные сплайны, а c_j –некоторые параметры, определяемые из задачи поиска минимума функционала

$$I = \int_0^1 [x^{\alpha} p(x) u'^2(x) + q(x)u^2(x) - 2f(x)u(x)]dx. \tag{5}$$

Будем строить приближённое решение вариационно-сеточным методом. [8] На промежутке $[\varepsilon, 1], \varepsilon > 0$ построим сетку узлов

$$x_j = \varepsilon + jh, \quad h = \frac{1-\varepsilon}{n},$$

где n - натуральное число.

Зададим базисные функции следующим образом:

$$\omega_j(x) = \begin{cases} \dfrac{x - x_{j-1}}{x_j - x_{j-1}}, x \in [x_{j-1}, x_j], \\ \dfrac{x - x_{j+1}}{x_j - x_{j+1}}, x \in [x_j, x_{j+1}], \\ 0, \quad x \notin [x_{j-1}, x_{j+1}] \end{cases}$$

Задача на минимум функционала (5) приводит к следующей системе уравнений:

$$\sum_{i=1}^{N} c_i g_{ij} = f_j, \quad j = 1, \ldots, N, \tag{6}$$

коэффициенты и правая часть которой вычисляются по формулам:

$$g_{ij} = \int_0^1 (x^\alpha \omega'_i(x)\omega'_j(x) + \omega_i(x)\omega_j(x))dx,$$

$$f_j = \int_0^1 f(x)\omega_j(x)dx.$$

3.1. Вычисление правой части

Правая часть вычисляется по формуле

$$f_j = \int_0^1 f(x)\omega_j(x)dx.$$

При подстановке заданных базисных функций,

$$\omega_j(x) = \begin{cases} \dfrac{x - x_{j-1}}{x_j - x_{j-1}}, x \in [x_{j-1}, x_j], \\ \dfrac{x - x_{j+1}}{x_j - x_{j+1}}, x \in [x_j, x_{j+1}], \\ 0, \quad x \notin [x_{j-1}, x_{j+1}] \end{cases}$$

получаем следующую формулу:

$$f_j = \int_{x_{j-1}}^{x_j} f(x)\frac{x - x_{j-1}}{x_j - x_{j-1}}dx + \int_{x_j}^{x_{j+1}} f(x)\frac{x - x_{j+1}}{x_j - x_{j+1}}dx. \qquad (1)$$

Строим приближение полученных интегралов (1) по составной формуле Симпсона: [7]

$$\int_a^b f(x)dx = \frac{b-a}{3n}(f_0 + 4(f_1 + f_3 + ... + f_{n-1}) + 2(f_2 + f_4 + ... + fn - 2) + f_n),$$

где n - количество частей, на которые разбит отрезок [a, b].

Программу для вычисления интеграла по формуле Симпсона с использованием параллельных секций [2] можно найти в приложении А.

3.2. Вычисление элементов матрицы

$$g = \begin{pmatrix} g_{00} & g_{01} & 0 & \dots & 0 \\ g_{10} & g_{11} & g_{12} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & g_{(n-1)(n-2)} & g_{(n-1)(n-1)} & g_{(n-1)n} \\ 0 & \dots & \dots & g_{n(n-1)} & g_{nn} \end{pmatrix}$$

Для вычисления элементов матрицы необходимо вычислить производные базисных функций:

$$\omega'_j(x) = \begin{cases} \dfrac{1}{x_j - x_{j-1}}, x \in [x_{j-1}, x_j], \\ \dfrac{1}{x_j - x_{j+1}}, x \in [x_j, x_{j+1}], \\ 0, \quad x \notin [x_{j-1}, x_{j+1}]. \end{cases}$$

При вычисление элементов g_{ij} можно различать два случая:

а) диагональные элементы g_{00}, g_{nn}, и остальные диагональные элементы g_{ii},

б) остальные элементы.

Заметим,что диагональные элементы вычисляются по формуле

$$g_{ii} = \int_0^1 (x^\alpha (\omega'_i(x))^2 + (\omega_i(x))^2) dx.$$

Если $|i - j| > 1$, то носители функций ω_i и ω_j не пересекаются, и наша матрица имеет трехдиагональный вид. Так как эта матрица еще и симметричная, то достаточно вычислить g_{ii+1}.

Тогда недиагональные элементы имеют вид:

$$g_{ii+1} = \int_{x_i}^{x_{i+1}} (x^\alpha \omega'_i(x)\omega'_{i+1}(x) + \omega_i(x)\omega_{i+1}(x)) dx,$$

$$g_{i-1i} = \int_{x_{i-1}}^{x_i} (x^\alpha \omega'_i(x)\omega'_{i-1}(x) + \omega_i(x)\omega_{i-1}(x)) dx.$$

4. Метод решения системы линейных уравнений

4.1. Метод прогонки

Для решения системы

$$Ax = f \tag{1}$$

с трехдиагональной матрицей наиболее часто применяется метод прогонки. [4]

$$A = \begin{pmatrix} b_0 & c_0 & 0 & \dots & 0 \\ a_1 & b_1 & c_1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & \dots & \dots & a_n & b_n \end{pmatrix}$$

Перепишем систему в следующем виде:

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = f_i, i = 1..n.$$

Метод прогонки основывается на предположении, что искомые неизвестные связаны рекуррентным соотношением:

$$x_i = \alpha_{i+1} x_{i+1} + \beta_{i+1}. \tag{2}$$

Отсюда получаем выражение для коэффициентов: $\begin{cases} \alpha_{i+1} = \frac{-c_i}{a_i \alpha_i + b_i}, \\ \beta_{i+1} = \frac{f_i - a_i \beta_i}{a_i \alpha_i + b_i} \end{cases}$ (3)

Так как значения a_0 и c_n равны 0, то $\begin{cases} \alpha_1 = \frac{-c_0}{b_0}, \\ \beta_1 = \frac{f_0}{b_0} \end{cases}$ (4)

После нахождения прогоночных коэффициентов α и β , используя рекуррентное соотношение(2), получим решение системы (1). При этом,

$$x_i = \alpha_{i+1} x_{i+1} + \beta_{i+1}, i = n-1, .., 1$$

$$x_n = \frac{f_n - a_n \beta_n}{b_n + a_n \alpha_n} \tag{5}$$

4.2. Метод встречной прогонки

Рассмотренный в предыдущем пункте метод прогонки, определяемый соотношениями (3)-(5), называют правой прогонкой. Аналогично можно выписать формулы для левой прогонки.

$$\xi_n = \frac{-a_n}{b_n}, \eta_n = \frac{f_n}{b_n}, \tag{6}$$

$$\xi_i = \frac{-a_i}{c_i\xi_{i+1} + b_i}, \eta_i = \frac{f_i - c_i\eta_{i+1}}{c_i\xi_{i+1} + b_i} \tag{7}$$

$$x_{i+1} = \xi_{i+1}x_i + \eta_{i+1}, i = 0, .., n-1 \tag{8}$$

$$x_0 = \frac{f_0 - c_0\eta_1}{b_0 + c_0\xi_1} \tag{9}$$

Метод встречной прогонки получается комбинированием левой и равой прогонки,и допускает распараллеливание на два потока. . [10]

Пусть $p = \frac{n}{2}$. Будем вычислять x_i,где $i \in [0, ...p]$, методом правой прогонки в первом потоке, а во втором потоке x_j, где $j \in [p+1, ...n]$, методом левой прогонки. Значение x_p находим из системы:

$$\begin{cases} x_p = \alpha_{p+1}x_{p+1} + \beta_{p+1}, \\ x_{p+1} = \xi_{p+1}x_p + \eta_{p+1} \end{cases}$$

Найдя указанное значение x_p, в первом потоке можно по формуле (5) найти все x_i, при $i \in [0, ...p]$, а во втором - по формуле (8) – все x_j, при $j \in [p+1, ...n]$. [6]

Реализацию параллельного алгоритма метода встречной прогонки с использованием технологии OpenMP [1] можно найти в приложении В.

5. Результаты

Вариационно-сеточный метод был применен к уравнению

$$(-x^{\alpha}u'(x))' + u(x) = \frac{x^{3-\alpha} - 2(3-\alpha)x - 1}{3-\alpha},$$

на сетке узлов $x_j = \varepsilon + j * \frac{1-\varepsilon}{100}$, при значениях $\varepsilon = 0.01, \alpha = 1, 1.5, 1.8, 2$. Ниже представлены графики, на которых изображено точное (красным цветом) и приближённое (синим цветом) решения. 5

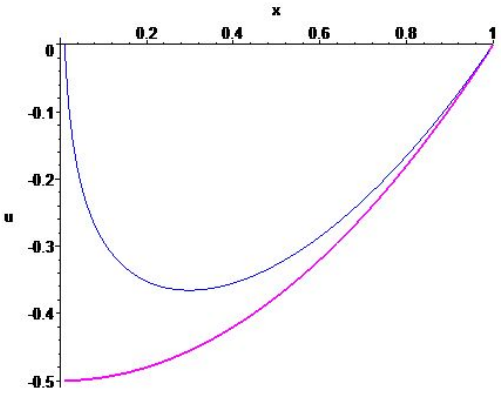

Рис. 1: при $\alpha = 1$

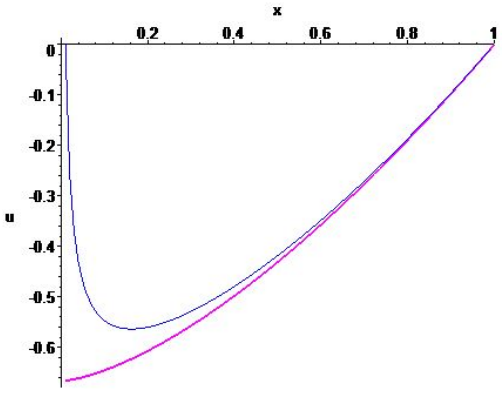

Рис. 2: при $\alpha = 1.5$

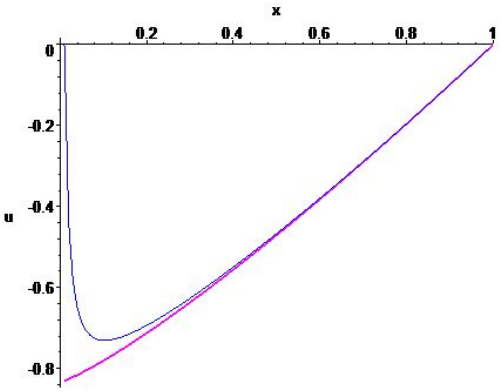

Рис. 3: при $\alpha = 1.8$

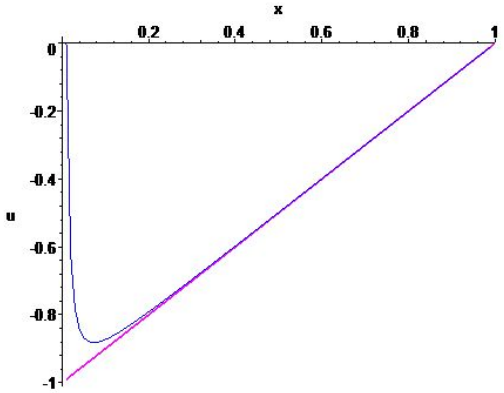

Рис. 4: при $\alpha = 2$

Заключение

- Были изучены методы решения краевой задачи вариационным методом.
- Разработан алгоритм решения краевой задачи с помощью сплайнов.
- Составлена программа решения краевой задачи.
- Получены графики, иллюстрирующие численное решение задачи, для различных значений параметров.

Список литературы

[1] Gerber Richard. Getting Started with OpenMP // Intel Developer Zone. — 2012. — Access mode: `https://software.intel.com/articles/getting-started-with-openmp/`.

[2] R. Andrews Gregory. Foundations of Multithreaded, Parallel, and Distributed Programming. — M.: Pearson, 2003.

[3] R. Courant. Variational methods for the solution of problems of equilibrium and vibrations. Bull.Amer.Math. Soc. — Наука, 1943.

[4] А. А. Абрамов В. Б. Андреев. О применении метода прогонки к нахождению периодических решений дифференциальных и разностных уравнений. Ж. вычисл. матем. и матем. физ. — 1963.

[5] Бурова И.Г. Демьянович Ю.К. Минимальные сплайны и их приложения. — С.-Петербургский университет, 2010.

[6] Демьянович Ю.К. Евдокимова Т.О. Чистяков П.П. Архитектура параллельных систем. — Издательство СПБГУ, 2014.

[7] Костомаров Д.П. Фаворский А.П. Вводные лекции по численным методам. Классический университетский учебник. — Логос, 2004.

[8] С.Г. Михлин. О сеточной аппроксимации вырождающихся одномерных дифферен циальных уравнений второго порядка. Вестн.ЛГУ №I. — 1973.

[9] С.Г. Михлин. Вариационно-сеточная аппроксимация. Зап. научн. сем. ЛОМИ. — Наука, 1974. — Режим доступа: `http://mi.mathnet.ru/znsl2786`.

[10] Самарский А.А. Николаев Е.С. Методы решения сеточных уравнений. Главная редакция физико-математической литературы. — Наука, 1978.

[11] Ю.К. Демьянович. Об аппроксимации минимальными сплайнами. — Ж. Проблемы математического анализа. Вып. 71., 2013.

A. Метод Симпсона

```
#include ''stdafx.h''
#include <stdio.h>
#include <omp.h>

double func(double x) //Функция f(x)
{
    double res = x;
    return res;
}

int main()
{
    const int n = 6;
    double a = 0.;
    double b = 1.;
    long float h = b/n;
    double x[n+1];
    double oddSum = 0;
    double evenSum = 0;
    double f[n+1];
    double integral = 0;
    for(int i = 0; i <=n; i++)
    {
        x[i] = a +i*h;
    }
    for(int i =0; i <=n; i++)
    {
        f[i] = func(x[i]);
    }
    #pragma omp parallel
    {
```

```
        #pragma omp sections //Будем параллельно вычислять
                             //каждую из сумм формулы Симпсона
        {
            #pragma omp section
            {
                for(int i = 1; i < n; i= i+2)
                {
                    oddSum = oddSum + f[i];
                }
            }
            #pragma omp section
            {
                for(int j =2; j<n; j= j+2)
                {
                    evenSum = evenSum + f[j];
                }
            }
        }
    }
    integral = ((b-a)/(3*n))*(f[0]+f[n]+4*oddSum + 2*evenSum);
    printf("Result = %.2f\n", integral );
}
```

В. Встречный метод прогонки

```
#include <iostream>
#include <omp.h>
#include <cstdlib>
using namespace std;
const int n;
double* a = new double[n];
double* b = new double[n];
double* c = new double[n];
double* f = new double[n];

void forright(int p, double *a, double *b, double *c, double *f,
    double *alpha, double *beta) //коэффициенты для правой прогонки
{
alpha[1] = -c[0] / b[0];
beta[1] = f[0] / b[0];
for (int i = 2; i <= p+1; i++)
{
alpha[i] = (-c[i - 1]) / (b[i - 1] + a[i - 1] * alpha[i - 1]);
beta[i] = (f[i - 1] - a[i - 1] * beta[i - 1]) / (a[i - 1] * alpha[i
}
}
void forleft(int p, double *a, double *b, double *c, double *f,
    double *xi, double *eta) //коэффициенты для левой прогонки
{
xi[n] = -a[n] / b[n];
eta[n] = f[n] / b[n];
for (int i = n-1; i > p; i--)
{
xi[i] = (-a[i])/(b[i] + c[i] * xi[i + 1]);
eta[i] = (f[i] - c[i] * eta[i + 1]) / (c[i] * xi[i + 1] + b[i]);
}
```

```
}
void right(int p,double *x, double *alpha, double *beta)
                                          //правая прогонка
{
for (int i = p - 1; i >= 0; i--)
{
x[i] = alpha[i + 1] * x[i + 1] + beta[i+1];
}
}
void left(int p, double *x, double *xi, double *eta)
                                          //левая прогонка
{
for (int i = p; i <= n-1; i++)
{
x[i+1] = xi[i + 1] * x[i] + eta[i + 1];
}
}
int main()
{
int p = (n+1) / 2;
double* x = new double[n];
double* alpha = new double[n];
double* beta = new double[n];
double* xi = new double[n];
double* eta = new double[n];
#pragma omp parallel sections
{
#pragma omp section
{
forright(p, a, b, c, f, alpha, beta);
}
#pragma omp section
{
```

```
forleft(p, a, b, c, f, xi, eta);
}
}
x[p] = (beta[p + 1] + alpha[p + 1] * eta[p+1]) / (1 - alpha[p + 1] *
#pragma omp parallel sections
{
#pragma omp section
{
right(p, x, alpha, beta);
}
#pragma omp section
{
left(p, x, xi, eta);
}
}
return 0;
}
```